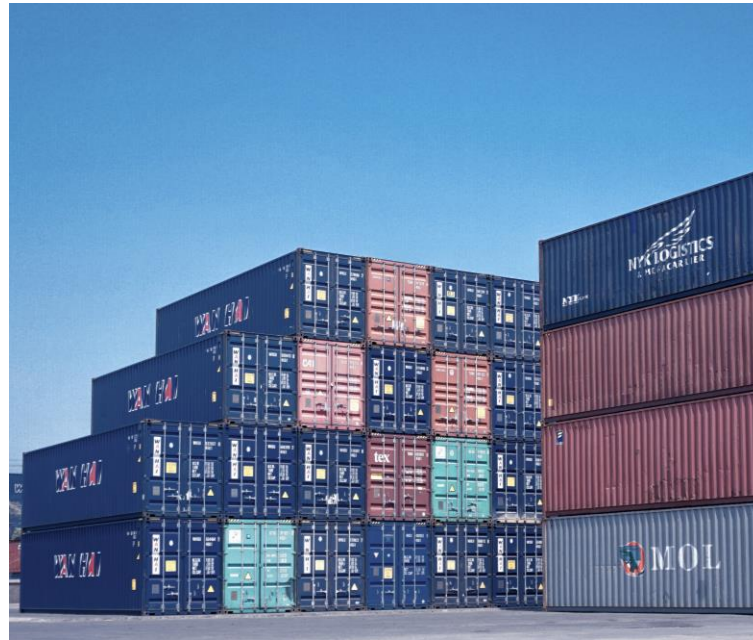


Microservices I: Designing Microservices

CNAE – Cloud Native Architecture & Engineering



Prof. Stefan Tai
Sebastian Werner, Elias Grünewald, Nicola Leschke, Maria Borges

Information Systems Engineering
TU Berlin

Admission Process

You can see if you are admitted by checking the ISIS page. There you will find a new category called "Assignments". If you have been admitted, you will find a questionnaire there titled "Participation Commitment" asking you to confirm your participation in the course by **today at 1:00 pm**.



Schedule

- Tuesday, 10:15 - 11:45, BIB 014
- Friday, 12:15 - 13:45 Uhr, HE 101

Kick-Off (for organisational questions) will be on Tuesday, Apr. 18 (virtually). The first lecture will be on Friday, Apr. 21 (HE 101).

Lectures, guest lectures and office hours will alternate. The exact schedule will follow here shortly.

For all virtual meetings, we will use the following Zoom-Link:

<https://tu-berlin.zoom.us/j/65284334867?pwd=QXBNV0xNb3ZNVmZ3SIRGZ21Uekg0QT09>

Meeting-ID: 652 8433 4867

Kenncode: 375255



Module Description Uploaded 14/04/23, 13:18



Schedule (V2) Uploaded 21/04/23, 11:18

▼ Assignments

🔒 Not available unless: You belong to **Zugelassen**



Participation Commitment

Closes: Tuesday, 25 April 2023, 1:30 PM

Please indicate if you are committed to participating in CNAE (you want to get graded in the course). Not answering until the deadline or rejecting the admission will make room for other students.

DEADLINE: 25.04.2023 13:00

🔒 Not available unless: You belong to **Zugelassen**

Learning Objectives

- Recapitulate what microservices are and how to distinguish microservice architectures from other architectural styles.
- Understand the characteristics of microservices and their implications.
- Apply systematic and principled practices to designing, implementing and deploying microservices and thereby resolving practical computing problems.

Agenda



Microservices I: Characteristics and Design Principles

1. Definitions
2. Characteristics
3. Design Principles
4. Summary

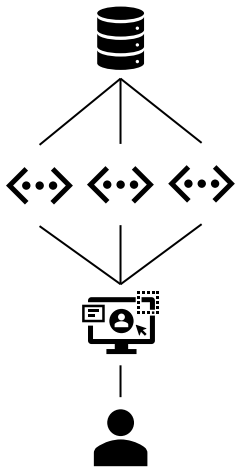
Microservices II: Using Microservices & Discussion

Intuitive Understanding?

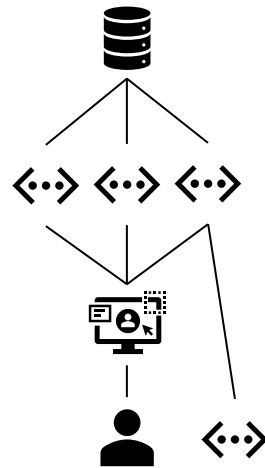
Which of the following architectures would you consider as a microservice architecture?



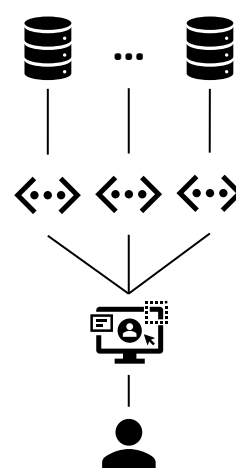
A



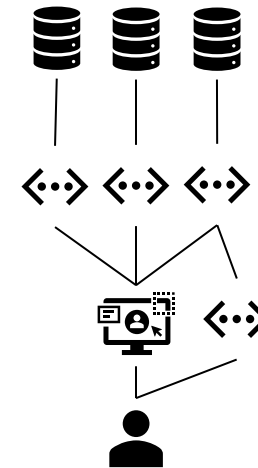
B



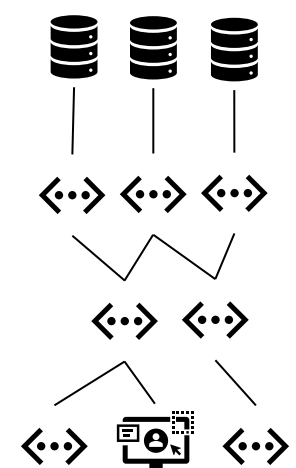
C



D



E



F

Microservice Architecture Definitions I

“In short, the microservice architectural style is an approach to developing a single application as a suite of **small [focused] services**, each **running in its own process and communicating with lightweight mechanisms**, often an HTTP resource API.

These services are **built around business capabilities and independently deployable** by **fully automated deployment** machinery.

There is a **bare minimum of centralized management** of these services, which may be written in different programming languages and use different data storage technologies.” [Martin Fowler]

Microservice Architecture Definitions II

“A *microservice* is an independently deployable component of bounded scope that supports interoperability through message-based communication. *Microservice architecture* is a style of engineering highly automated, evolvable software systems made up of capability-aligned microservices.”

[Nadareishvili et al. (2016) – Microservice Architecture]

“Loosely coupled, service-oriented architecture with bounded contexts”

[Adrian Cockcroft]

“Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs. These services are owned by small, self-contained teams.”

[<https://aws.amazon.com/microservices/>]

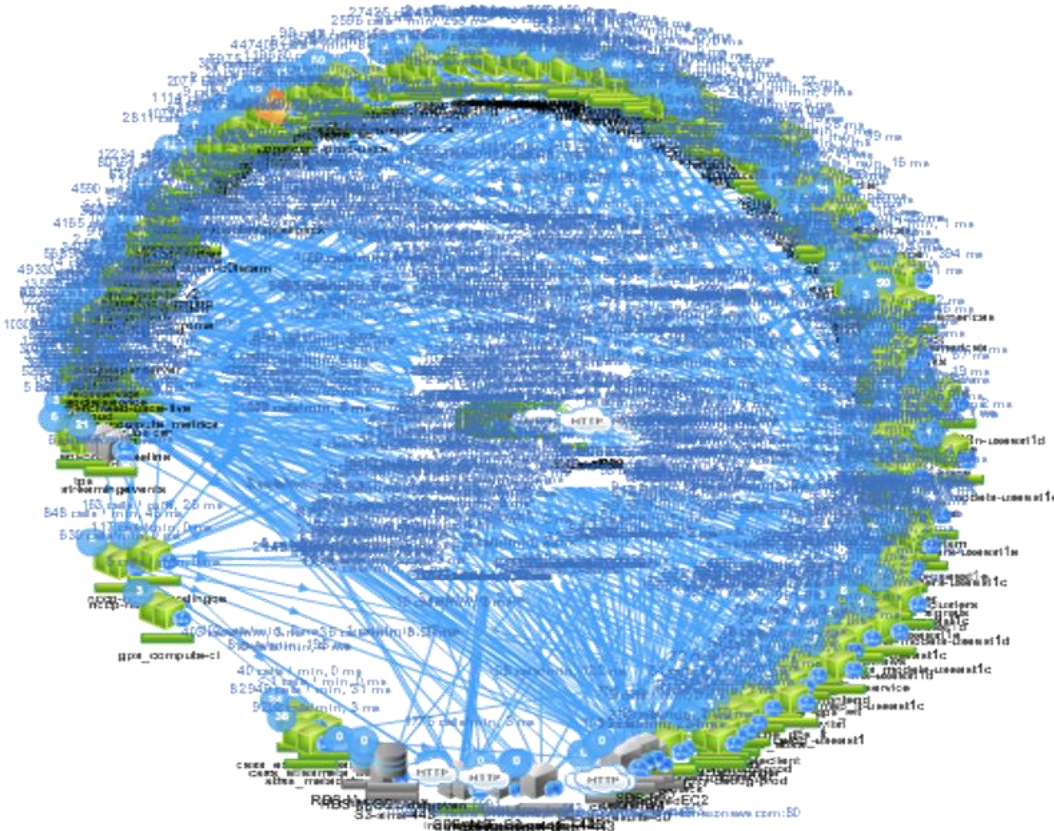
Example: Microservices at Netflix aka the Death (Dev) Star

Separate Datastore for each service

- No interference between teams on the same database
- Master Data Management to fix inconsistencies

Communication over well-defined service-interfaces

Servers are cattle, not pets



What is a service interface?

What kinds of interfaces do you connect with microservices?

“A service interface is a published interface used to invoke a service.”

[James McGovern et al. 2003]

Interface Maxims



The Mars Climate Explorer



Documentation matters.

- APIs should be easy to use and hard to misuse.
- Structure requirements as use-cases.
- API design is not a solitary activity.
- When in doubt, leave it out.
- Use consistent parameter ordering across methods.
- ...

Find out more in
Microservices II



Bloch, Joshua. "How to design a good API and why it matters." *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. 2006.

Distinguishing Microservices from other Architectures



What other architectural styles are you familiar with?

How can they be distinguished from Microservices?

Monolith

n-Tier Architecture

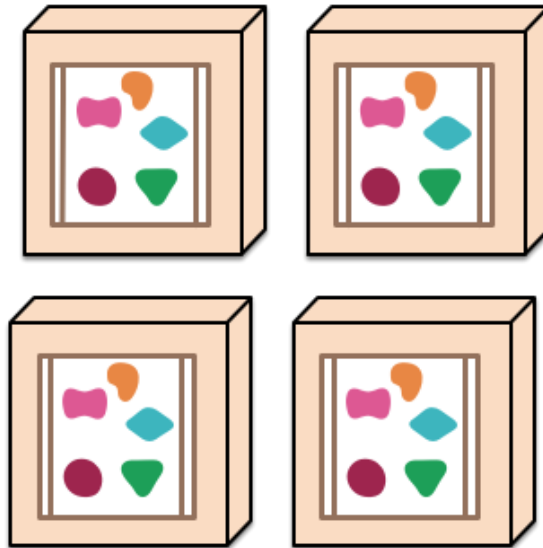
Service-Oriented Architecture

Monoliths versus Microservices

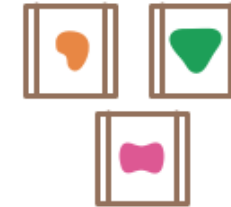
A monolithic application puts all its functionality into a single process...



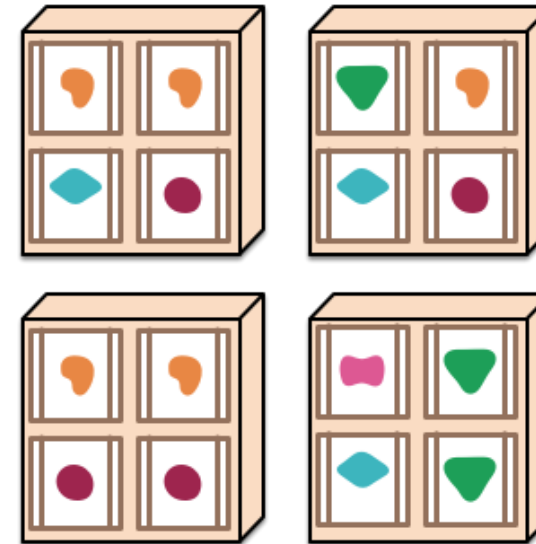
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.



Service Oriented Architecture versus Microservices

SERVICE-ORIENTED ARCHITECTURE	MICROSERVICES ARCHITECTURE
Maximizes application service reusability	Focused on decoupling
A systematic change requires modifying the monolith	A systematic change is to create a new service
DevOps and Continuous Delivery are becoming popular, but are not mainstream	Strong focus on DevOps and Continuous Delivery
Focused on business functionality reuse	More importance on the concept of “ bounded context ”
For communication it uses Enterprise Service Bus (ESB)	For communication uses less elaborate and simple messaging systems
Supports multiple message protocols	Uses lightweight protocols such as RESTful HTTP, Thrift APIs
Use of a common platform for all services deployed to it	Application Servers are not really used, it's common to use cloud platforms
Use of containers (such as Docker) is less popular	Containers work very well with microservices
SOA services share the data storage	Each microservice can have an independent data storage
Common governance and standards	Relaxed governance, with greater focus on teams collaboration and freedom of choice

Source: <https://dzone.com/articles/microservices-vs-soa-is-there-any-difference-at-all>

Example: Amazon design rules (Jeff Bezos Mandate)

1. All teams will henceforth expose their data and functionality through **service interfaces**.
2. Teams must communicate with each other through these interfaces.
3. There will be **no other form of inter-process communication allowed**:
 - no direct linking, no direct reads of another team's data store,
 - no shared-memory model,
 - no back-doors whatsoever.

The only communication allowed is via service interface calls over the network.
4. It **doesn't matter what technology they use**. HTTP, Corba, Pubsub, custom protocols -- doesn't matter.
5. All service interfaces, without exception, must be designed from the ground up to be **externalizable**. That is to say, the team must plan and design to be able to **expose the interface to developers in the outside world**. No exceptions.
6. Anyone who doesn't do this will be fired.
7. Thank you; have a nice day!

Most services are not externally available

Service depth?



Agenda



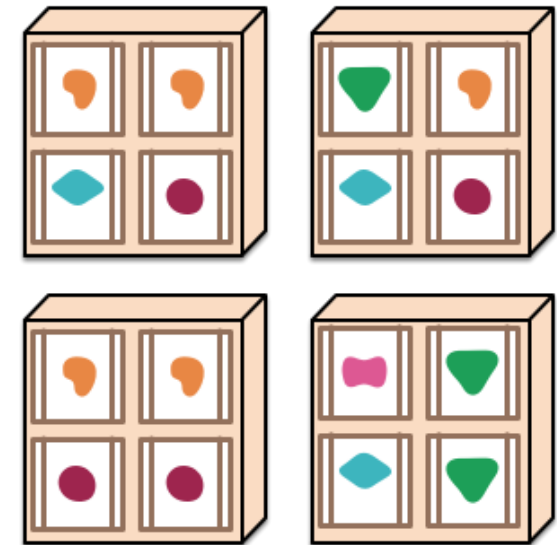
Microservices I: Characteristics and Design Principles

1. Definitions
2. Characteristics
3. Design Principles
4. Summary

Characteristics of Microservices

[Fowler]

1. Componentization via Services
2. Organized around Business Capabilities
3. Products not Projects
4. Smart endpoints and dumb pipes
5. Decentralized Governance
6. Decentralized Data Management
7. Infrastructure Automation
8. Design for failure
9. Evolutionary Design



Material adopted from: <https://martinfowler.com/articles/microservices.html>

1. Componentization via Services

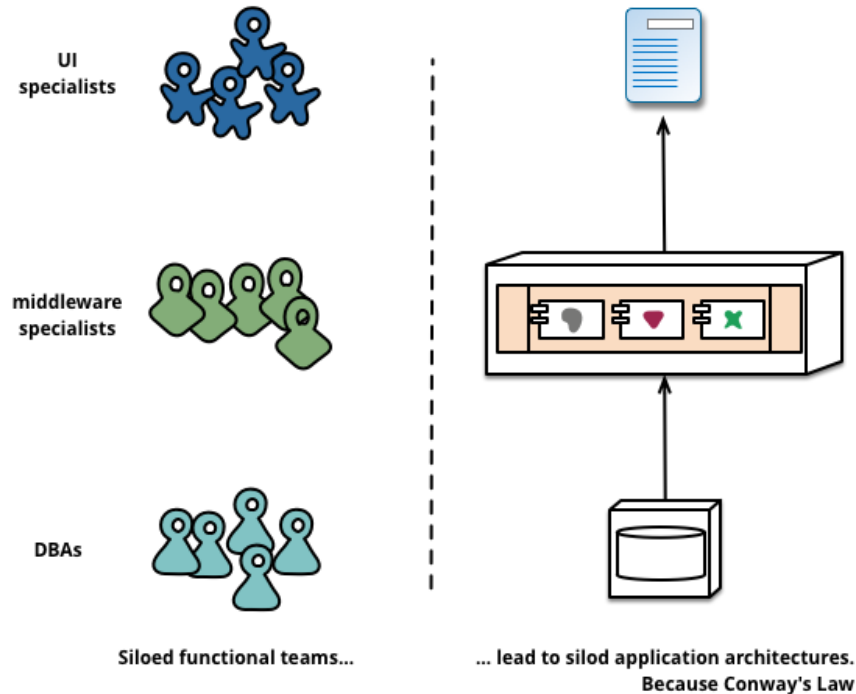
A **component** is a unit of software that is independently replaceable and upgradeable.

Microservice architectures will use libraries, but their primary way of componentizing their own software is by breaking down into **services**.

We define libraries as components that are linked into a program and called using in-memory function calls, while **services are out-of-process components who communicate with a mechanism** such as a web service request, or remote procedure call.

2. Organized around Business Capabilities (Conway's Law)

Siloed functional team lead to siloed application architectures.

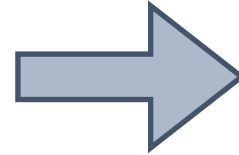


„Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.“

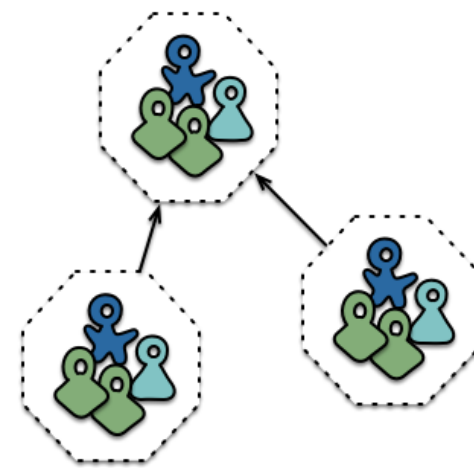
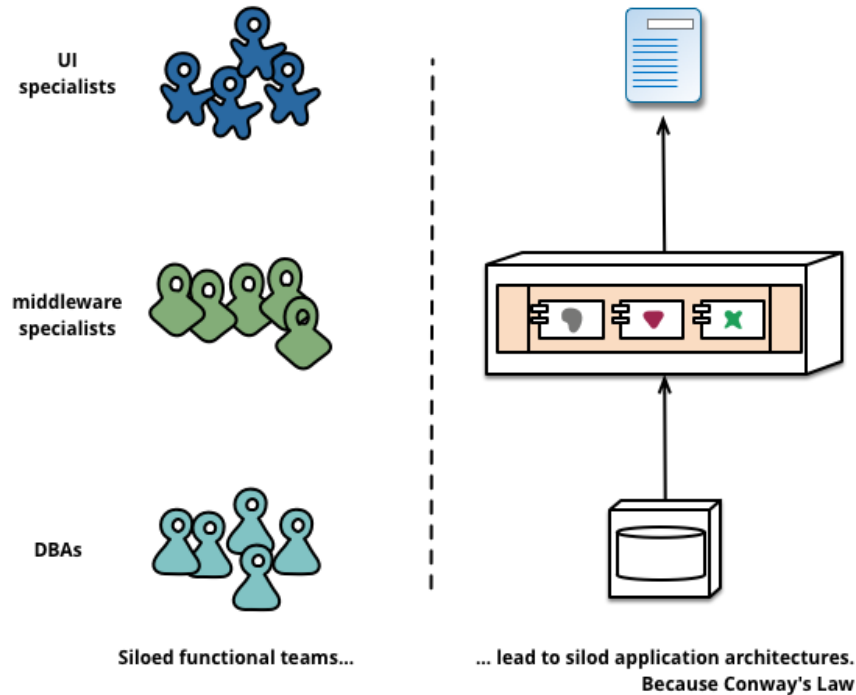
Conways Law
Mel Conway (1968) "How Do Committees Invent?". In: *Datamation*.

2. Organized around Business Capabilities

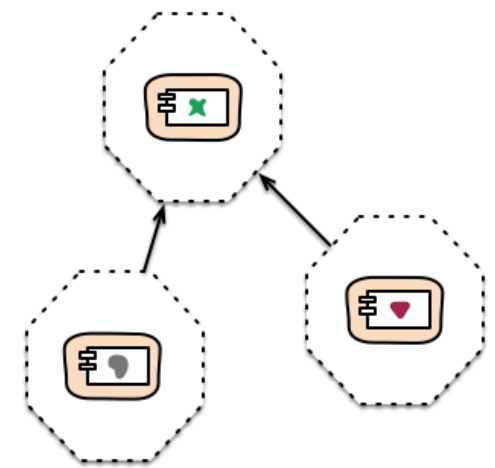
Siloed functional team lead to siloed application architectures.



Cross-functional teams organize around capabilities.



Cross-functional teams...

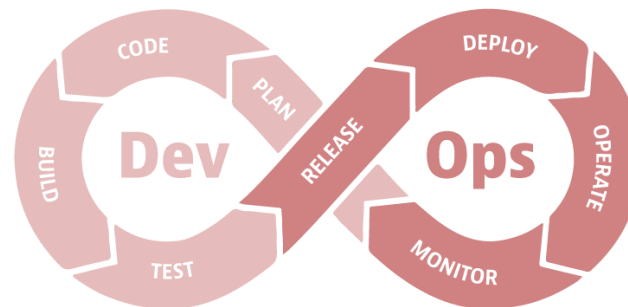


... organised around capabilities
Because Conway's Law

3. Products, not Projects

Project model: the aim is to deliver some piece of software which is then considered to be completed. On completion the software is handed over to a maintenance organization and the project team that built it is disbanded.

Product model: A development team takes full responsibility for the software in production (“you build it, you run it”).



Additional Literature: Len Bass, Ingo Weber, Liming Zhu, Addison-Wesley. “DevOps: A Software Architect's Perspective.” Professional, 2015.

4. Smart endpoints and dumb pipes

Applications built from microservices aim to be **as decoupled and as cohesive as possible**

- they own their own domain logic and act more as filters in the classical Unix sense
- receiving a request, applying logic as appropriate and producing a response.

These are choreographed using simple REST-“ish” protocols rather than complex protocols such as WS-Choreography or BPEL or orchestration by a central tool.

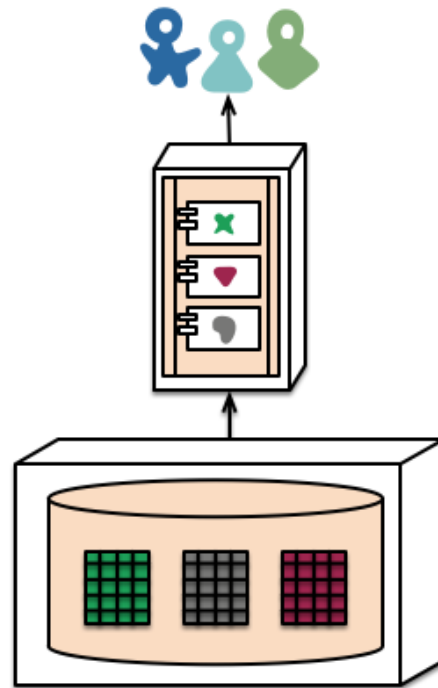
5. Decentralized Governance

One of the consequences of centralized governance is the tendency to standardize on **single technology platforms**. Experience shows that this approach is constricting - not every problem is a nail and not every solution is a hammer.

Rather than use a set of defined standards written down somewhere on paper they prefer the idea of **producing useful tools that other developers can use to solve similar problems** to the ones they are facing.

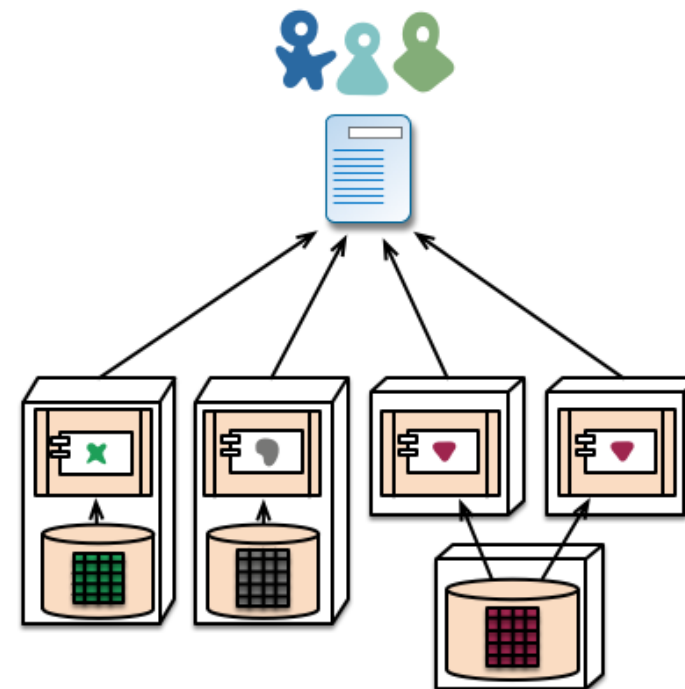
6. Decentralized Data Management

In monoliths, the application uses a shared database.



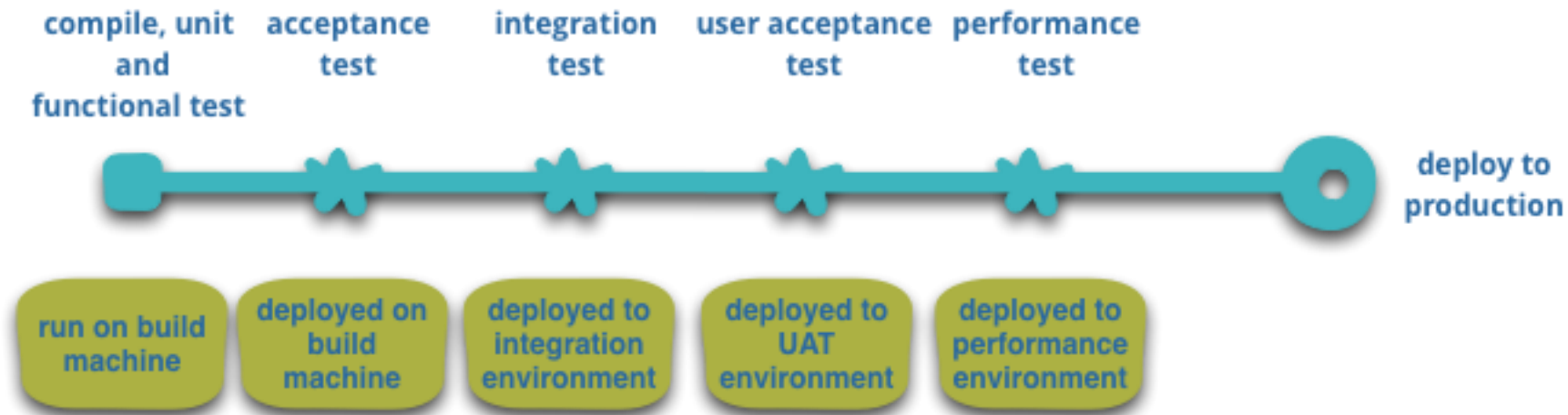
monolith - single database

Each microservice maintains its own database.



microservices - application databases

7. Infrastructure Automation



Automated, continuous deployment and testing processes

8. Design for failure

A consequence of using services as components, is that applications need to be designed so that they can tolerate the failure of services.

Any service call could fail due to unavailability of the supplier, the client has to respond to this as gracefully as possible. This is a disadvantage compared to a monolithic design as it introduces additional complexity to handle it.

The consequence is that microservice teams **constantly reflect on how service failures affect the user experience.**

9. Evolutionary Design

The key property of a component is the notion of **independent replacement and upgradeability**.

You want to keep things that change at the same time in the same module. Parts of a system that change rarely should be in different services to those that are currently undergoing lots of churn.

Putting components into services adds an **opportunity for more granular release planning**. With a monolith any changes require a full build and deployment of the entire application. With microservices, however, you only need to redeploy the service(s) you modified. This can simplify and speed up the release process.

Agenda



Microservices I: Characteristics and Design Principles

1. Definitions
2. Characteristics
3. Design Principles
4. Summary

Ideal Design Principles I

Microservice design principles combine well-known modular software design guidelines (e.g. Unix design maxims) and best-practice experience in building SOA at scale (e.g. Netflix, Amazon Web Services, SoundCloud).

A microservice architecture is domain-specific, i.e. bounded contexts of microservices must be chosen depending on data models, reflecting the scope of business capabilities appropriately.

Microservices are aligned to (individual) product ownership responsibilities

- Tooling and frameworks for development: should depend on individual team capabilities and preferences; implement own tooling, if appropriate
- Runtime configuration: owners test and manage release configurations for deployed artifacts themselves
- Observability: custom monitoring and debugging

Ideal Design Principles II

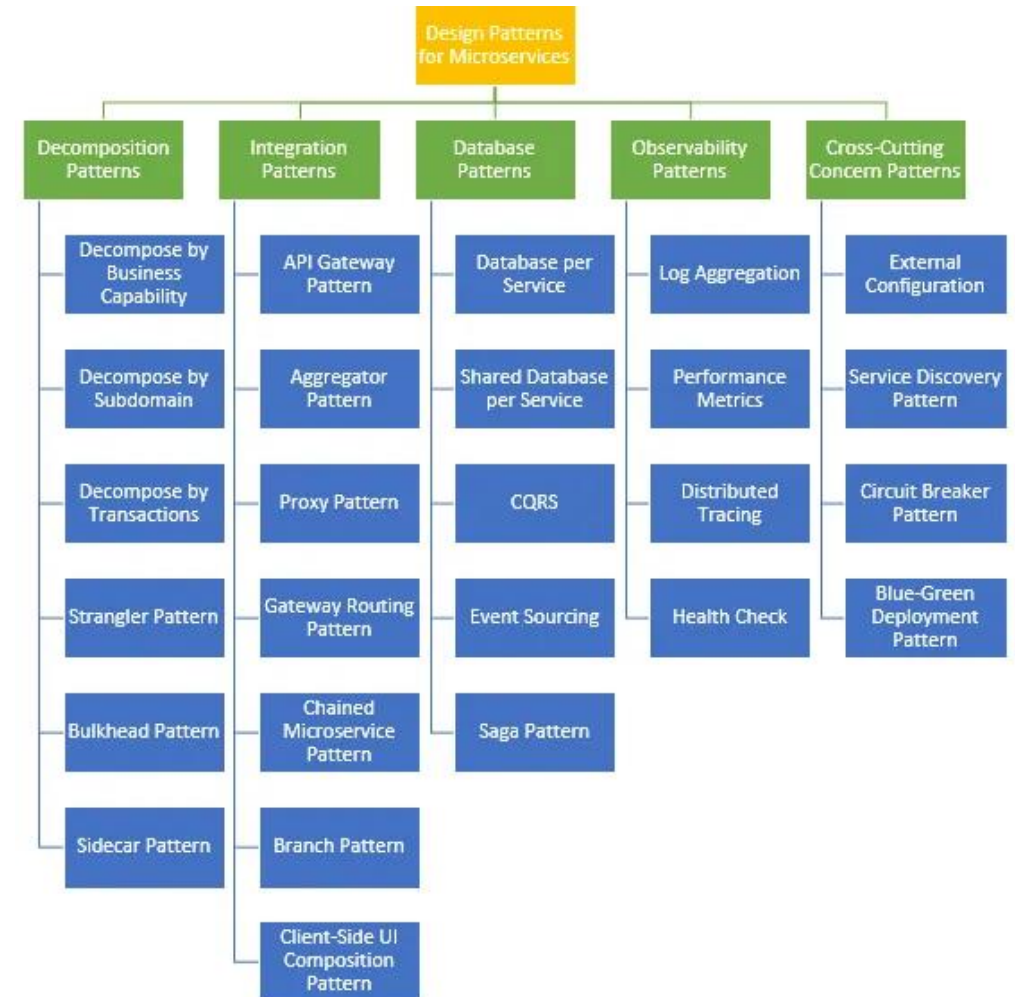
Microservice architectures adopt (shared) platforms in their development, deployment and runtime environment

- Infrastructure: Rely on managed, on-demand provisioning of virtualized hardware resources, e.g. Cloud-IaaS, container-based virtualization
- Use tools for automation for code versioning, testing and deployment
- Data stores: individual instances (or shards) per microservice, but shared best practices and technology
- Service orchestration: interoperability through the adoption of common discovery and loadbalancing tools
- Shared security and identity management
- Company policies, e.g. logging to central auditing platform

From Qualities to Design Patterns

Find out more in the upcoming lectures!

Availability
Consistency
Decentralized governance
Decoupled
Flexibility
Independent, autonomous
Resilience
Scalability
Security
...



<https://medium.com/@madhukaudantha/microservice-architecture-and-design-patterns-for-microservices-e0e5013fd58a>

From Qualities to Design Patterns

Availability
 Consistency
 Decentralized governance
 Decoupled
 Flexibility
 Independent, autonomous
 Resilience
 Scalability
 Security



<https://medium.com/@madhukaudantha/microservice-architecture-and-design-patterns-for-microservices-e0e5013fd58a>

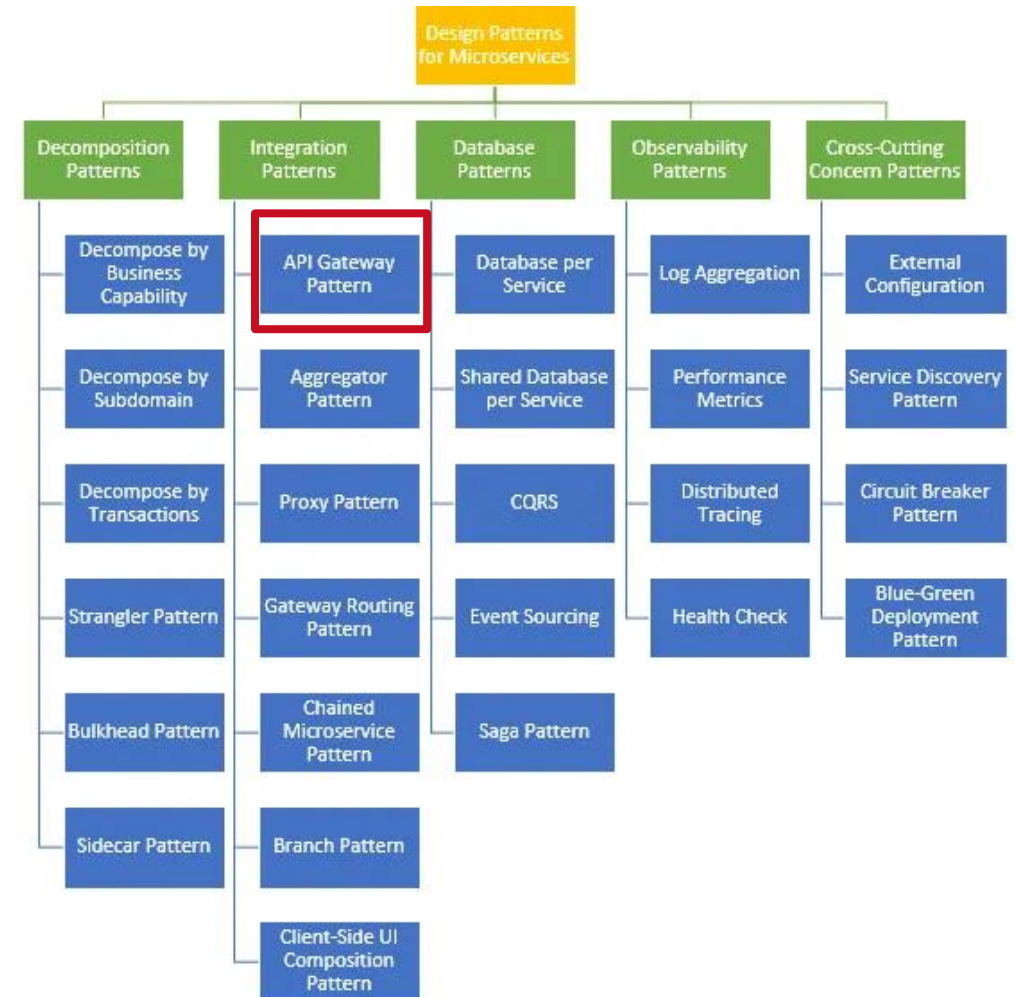
Decomposition Patterns



What is the difference between a decoupled and a distributed architecture?

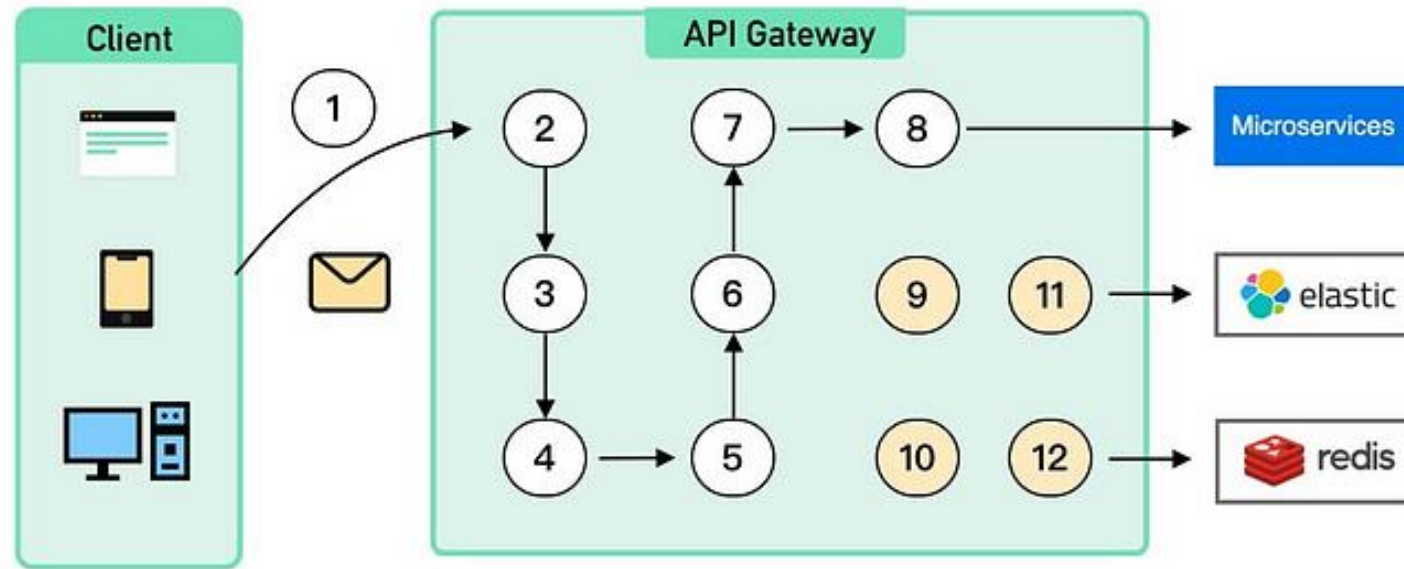
From Qualities to Design Patterns

Availability
 Consistency
 Decentralized governance
 Decoupled
 Flexibility
 Independent, autonomous
 Resilience
 Scalability
 Security



<https://medium.com/@madhukaudantha/microservice-architecture-and-design-patterns-for-microservices-e0e5013fd58a>

API Gateway Pattern



API Gateway is one of the essential Microservices pattern which is used to provide **a single entry point for external consumers** to access the services. It acts as a reverse proxy and routing layer, which is **responsible for request routing, composition, and protocol translation**, among other things.

<https://medium.com/javarevisited/50-microservices-interview-questions-for-java-programmers-70a4a68c4349>

What are advantages of the API gateway pattern?

- decouple the external consumers from the internal implementation
- provide a single entry point for external consumers:
 - simplifies the client-side service discovery
 - reduces the number of network calls
- address cross-cutting concerns:
 - Security, rate limiting, caching
- aggregate multiple services into a single response
- protocol and content type translations

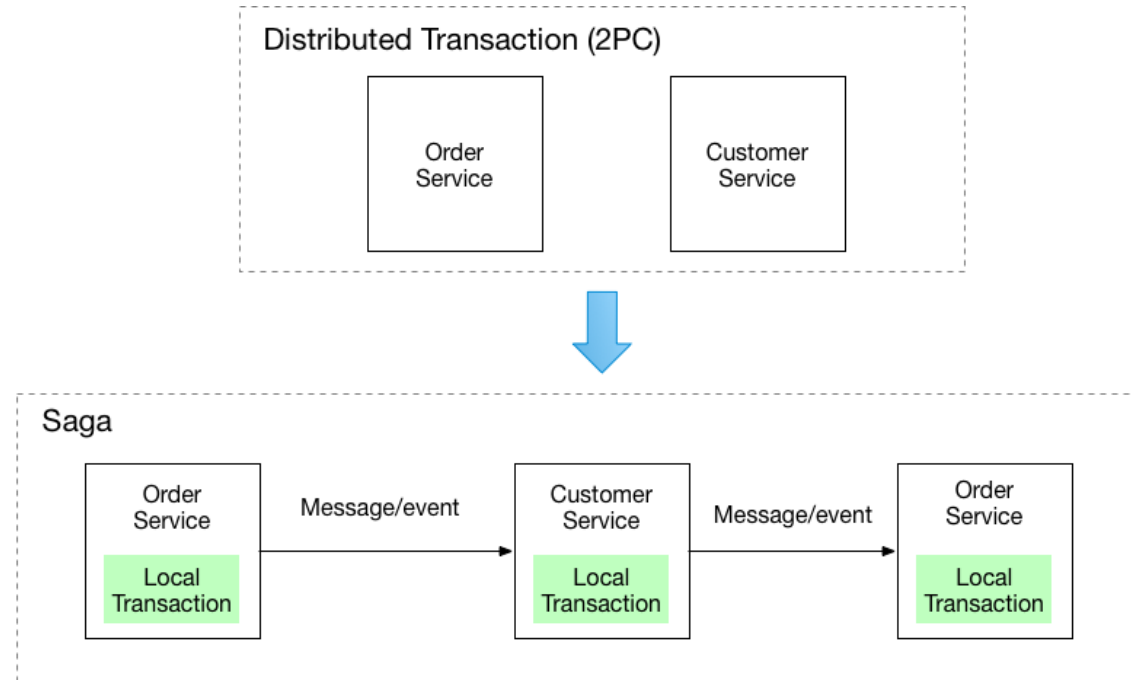
From Qualities to Design Patterns

Availability
 Consistency
 Decentralized governance
 Decoupled
 Flexibility
 Independent, autonomous
 Reliability
 Resilience
 Scalability
 Security



<https://medium.com/@madhukaudantha/microservice-architecture-and-design-patterns-for-microservices-e0e5013fd58a>

SAGA Pattern



There are two ways of coordination sagas:

- Choreography - each local transaction publishes domain events that trigger local transactions in other services
- Orchestration - an orchestrator (object) tells the participants what local transactions to execute

From Qualities to Design Patterns

Availability
 Consistency
 Decentralized governance
 Flexibility
 Independent, autonomous
 Reliability
 Resilience
 Scalability
 Stability



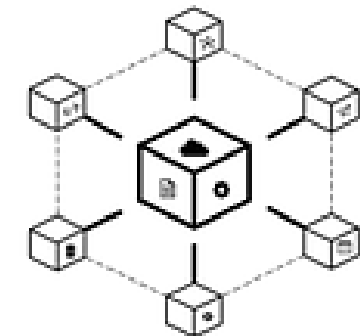
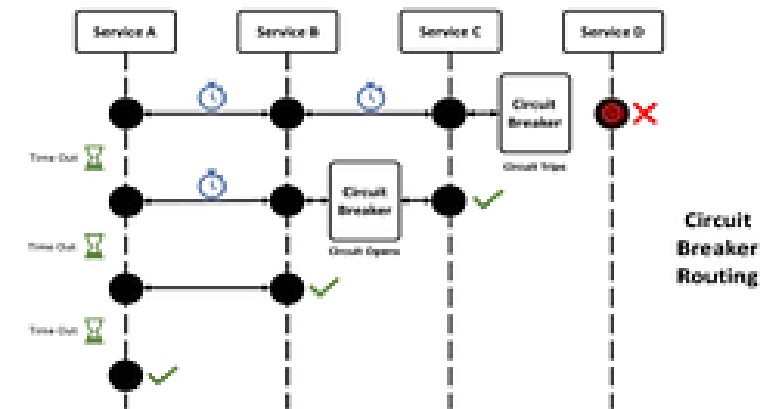
<https://medium.com/@madhukaudantha/microservice-architecture-and-design-patterns-for-microservices-e0e5013fd58a>

Circuit Breaker

The Circuit Breaker prevents the invocation of failed services by acting as a proxy for services that might fail.

There are three states:

- Closed
- Opened
- Half-opened



Circuit Breaker In Microservices

<https://medium.com/javarevisited/what-is-circuit-breaker-in-microservices-a94f95f5e5ae>

Agenda



Microservices I: Characteristics and Design Principles

1. Definitions
2. Characteristics
3. Design Principles
4. Summary

Summary: Microservices

...are **autonomously deployable services** providing a **focused amount** of functionality.

...are designed around the goal of enabling **fast changes**, both **incremental** and radical.

...are **standardized with respect to interfaces and deployment** procedures, but not things „below the surface“ (programming language, webserver, database, ...).

...require **organizational structure** and a **DevOps** culture.

- Small teams develop small services and own them as products
- Coordination through service interfaces
 - Changes are communicated, implemented, and added to the interface

What are the benefits and drawbacks of a Microservices architecture?

Pros

- Flexibility
- Scalability
- Resilience
- Agility
- Reusability
- ...

Cons

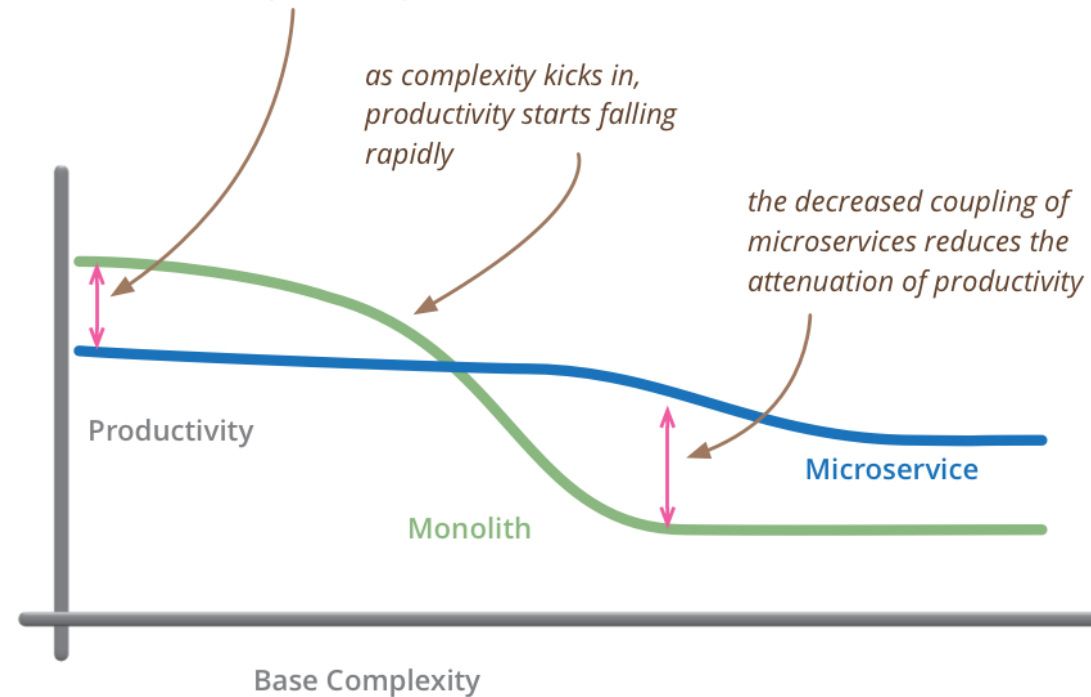
- complexity
 - testing and debugging
 - monitoring and management
- dealing with failure
- inter-service communication
- security
- eventual consistency
- ...

MicroservicesPremium

The increased complexity introduced through microservices is referred to as microservice premium.

The majority of software systems should be built as a single (modular) monolithic application.

for less-complex systems, the extra baggage required to manage microservices reduces productivity



but remember the skill of the team will outweigh any monolith/microservice choice

Source: <https://martinfowler.com/bliki/MicroservicePremium.html>

CNAE Schedule (preliminary)

